Per Monitor DPI

Developer Guide

# Introduction

This developer guide provides developer info on how to test your WPF application with our Per Monitor DPI preview. This preview build of WPF can be thought of as .NET 4.6.1 + the Per Monitor DPI feature.

To request access to the Per Monitor DPI preview build, see the “Per Monitor DPI Support” section of our [WPF 4.6.1 announcement blog post](http://blogs.msdn.com/b/wpf/archive/2015/10/29/wpf-in-net-4-6-1.aspx).

# Installation Steps

1. Follow the installation steps from the email invite that gave you access to the preview build.
2. You’ll also need to update the reference assemblies with the new APIs that the preview build has, you’ll need to do the following from an admin command prompt:

copy PresentationFramework.dll & PresentationCore.dll from %windir%\Microsoft.NET\Framework\v4.0.30319\WPF to the latest framework version folder under C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\4.6 or 4.6.1 [the latest one you have]

# Operating System Prerequisites

In order to enable Per Monitor DPI Awareness in your app, you should be running Windows 8.1 or Windows 10.

# Enabling Per Monitor DPI for your app

## Turn on Windows level Per monitor DPI awareness in app.manifest

WPF apps are System DPI aware by default, and need to declare themselves to be Per Monitor DPI aware via an app.manifest file. To add an app.manifest:

* Right click on your project folder, click Add -> New Item
* Click on General in the left pane, and chose Application Manifest File.
* Uncomment the following snippet of xml:

<application xmlns="urn:schemas-microsoft-com:asm.v3">

<windowsSettings>

<dpiAware xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true/PM</dpiAware>

</windowsSettings>

</application>

(Make sure it is set to “true/PM” where PM stands for Per Monitor)

## Turn on WPF Feature in App.Config

Today, you’ll also need to add the following snippet to app.config (inside <configuration> element):

<runtime>

<AppContextSwitchOverrides value = "Switch.System.Windows.DoNotScaleForDpiChanges=false"/>

</runtime>

* Yes, set it to false. Double negative FTW!
* Note: the AppContextSwitchOverrides cannot be set twice. If your application already has one set, you must semicolon delimit this switch inside the value attribute.

# Test Environment

Run on a PC with 2 monitors, running Windows 8.1 or Windows 10.

Set the 2 monitors DPI so that they are different (DPI of 100 on one and 150 on the other).

# Once Enabled

Doing the above steps will enable any WPF app, **with pure WPF content**, to work seamlessly as a Per Monitor DPI Aware app.

There are several scenarios that will take some additional coding in your application with newly available APIs:

## WPF Apps

* **Images** - often it is best to have different scales of images for different DPIs – see PerMonitorDpi\ImageScaling sample on github.com/WpfSamples.

We’re considering doing more work in the image scaling space, before release, to improve this scenario.

* **Hosted Hwnds or WindowsForms controls** - if your application uses HwndHost or WindowsFormsHost, you’ll want to listen to DpiChanged on that control and adapt your scaling as appropriate - see PerMonitorDPI\WinFormsHost sample on github.com/WpfSamples
* **RenderTargetBitmap** – If you are rendering a set of visuals, which are not part of the main visual tree, via RenderTargetBitmap – WPF should be told the target DPI for that set of visuals. Call VisualTreeHelper.SetRootDpi(rootVisual, dpiScaleInfo) before calling Measure and/or RenderTargetBitmap.
* **TextFormatting API calls** – if your application uses any System.Windows.Media.-TextFormatting APIs, you’ll need to pass in your target DPI into new constructor overrides or properties. When DPI changes, you’ll need to force a re-render of your text – see PerMonitorDpi\TextFormatting and PerMonitorDpi\FormattedTextExample samples on github.com/WpfSamples

## HWND Apps (NOT SUPPORTED)

The scenario where a Win32 hosts WPF via HwndSource does not currently support Per Monitor DPI. DPI changed messages (WM\_DPICHANGED) are only sent to the top level window, so WPF is not informed of a change. We’ll consider doing work in the future here, but it likely will require additional features from the Windows team.

## WindowsForms Apps (NOT SUPPORTED)

The scenario where a WindowsForms app hosts WPF via ElementHost does not currently support Per Monitor DPI. We’ll consider doing work in the future here, but it likely will require additional features from the Windows team.

# Using new APIs requires Reference Assembly Install Step

If you need to do more advanced coding, involving this feature, you’ll need to make sure that your app is targetting the same .NET Version that you updated the reference assemblies in (step 2 of installation steps in this document). Otherwise, Visual Studio and the compilers won’t be able to understand any of the new APIs.

# API Details

## Events and Virtuals

1. Window.DpiChanged event

as a window is moved to a different DPI monitor, this event will fire.

1. Image.DpiChanged event

as a window is moved to a different DPI monitor, this event will fire on all Images inside of that window. It is a routed event, so you could listen to it centrally, on the root element of each xaml file.

1. WindowsFormHost.DpiChanged event

as a window is moved to a different DPI monitor, this event will fire on all WindowsFormHosts inside of that window. It is a routed event, so you could listen to it centrally, on the root element of each xaml file.

1. HwndHost.DpiChanged event

as a window is moved to a different DPI monitor, this event will fire on all HwndHosts inside of that window. It is a routed event, so you could listen to it centrally, on the root element of each xaml file.

1. HwndSource.DpiChanged event

Most developers will be using the HwndSource that Window provides for them. If your application uses a HwndSource directly, you may find the following API useful:

DpiChanged event is fired on the HwndSource.

Note: During your event handler, if you mark this event as handled, WPF will not scale the UI for you, or notify the visuals of any DPI change.

1. Visual.OnDpiChanged virtual method

Each visual has a virtual method which can be overridden in order to listen to DPI change notifications on each visual. If your control needs to understand DPI for your rendering, you should override this method and behave appropriately.

protected virtual void OnDpiChanged(DpiScaleInfo oldDpi, DpiScaleInfo newDpi)

## VisualTreeHelper APIs

### VisualTreeHelper.GetDpi

/// <summary>

/// Gets the current DPI at which this visual is rendered.

/// </summary>

public static DpiScaleInfo GetDpi(Visual visual)

### VisualTreeHelper.SetRootDpi

/// <summary>

/// This method updates the DPI information of a visual.

/// It can only be called on a Visual with no parent.

/// </summary>

public static void SetRootDpi(Visual visual, DpiScaleInfo dpiInfo)

## TextFormatting Scenarios

If you use low level Text Formatting APIs and use custom TextSource, TextRun, GlyphRun, FormattedText to draw text in your app, then you need to use the following guidelines:

1. All text should be re-rendered when DPI is changed on the owner window.
2. TextSource, TextRunProperties, GlyphRun and FormattedText have a new PixelsPerDip property, which should be updated to reflect the DPI at which the text should be rendered. For guidelines on how to do this effectively, see PerMonitorDPI\FormattedTextExample and PerMonitorDPI\TextFormatting samples on github.com/WpfSamples.

**Known Issue:** DisplayMode text does not render properly on a DPI other than the DPI of the primary monitor. We are currently testing a fix for this, and will provide the fix in the next preview release.

# Communicating Feedback

Please let us know how this is working for you, or if you find any problems with the feature.

Email **wpfteam@microsoft.com**

We will update samples and this document as needed, look for updates.